COSC 264 Assignment

Andreas Willig
Dept. of Computer Science and Software Engineering
University of Canterbury

July 27, 2020

---

1 Administrivia

This assignment is part of the COSC 264 assessment process. It is worth 10% of the final marks. It centers around socket programming using the Python programming language. Your program should be a text mode program that can run from the command line.

Please use the "Question and Answer Forum" on the Learn platform for raising and discussing any unclear technical issues. Please **do not** send emails with technical questions directly to me or the tutors, instead use the learn forum. This way other people can benefit from the question (and the answer).

2 Plagiarism Warning

Your submissions are logged and originality detection software will be used to compare your solution with other solutions. Dishonest practice, which includes

- letting someone else create all or part of an item of work,
- copying all or part of an item of work from another person with or without modification, and
- allowing someone else to copy all or part of an item of work,

may lead to partial or total loss of marks, no grade being awarded, the failing grade 'X' (dishonesty) being awarded, or other serious consequences including notification of the University Proctor.

You are encouraged to discuss the general aspects of a problem with others. However, anything you submit for credit must be entirely your own work and not copied, with or without modification, from any other person. If you need help with specific details relating to your work, or are not sure what you are allowed to do, contact your tutors or lecturer for advice. If you copy someone else's work or share details of your work with anybody else, you are likely to be in breach of university regulations and the Computer Science and Software Engineering department's policy. For further information please see

- Academic Integrity Guidance for Staff and Students
  `www.canterbury.ac.nz/ucpolicy/GetPolicy.aspx?file=Academic-Integrity-Guidance-For-Staff-And-Students.pdf`
- Academic Integrity and Breach of Instruction Regulations in the University Calendar
  `www.canterbury.ac.nz/regulations/general-regulations/academic-integrity-and-breach-of-instruction-regulations/`

You will have to sign a plagiarism declaration upon submission of your assignment report.

3 Problem Description

Your task is to write two programs in Python. The first one, called `server` will allow the other program, called `client`, to ask the server for the current date or time of day. The server offers to deliver these in three different languages.

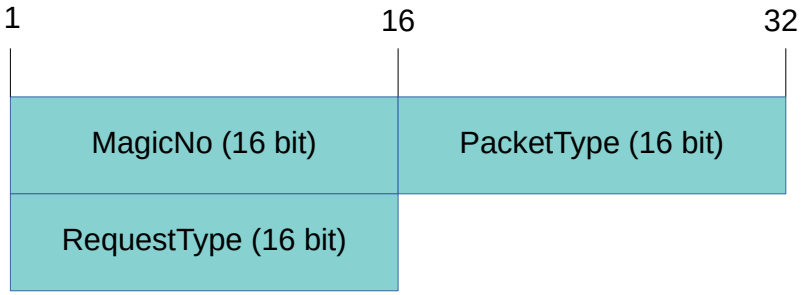
3.1 Packet Types and Packet Processing

The programs will require two different types of packets to be used: `DT-Request` and `DT-Response` packets (where 'DT' stands for "DateTime"). With a `DT-Request` a client requests either the date or the current time of day from the server. With a `DT-Response` packet the server returns both the date and current time of day in a binary representation, followed by either the date or the time of day (depending on the client's choice) in a textual representation.

These packets contain a number of 16- or 32-bit wide fields. All these fields shall use the big-endian format.

3.1.1 `DT-Request` packet

The format of the `DT-Request` packet is shown in the following figure:



It consists of the following fields:

- The first 16 bit field 'MagicNo' is taken up by a magic number, which needs to have the value `0x497E`. This is a simple safeguard to check whether a packet actually belongs to our Date-/Time protocol.
- The next 16 bit field 'PacketType' indicates the packet type within our DateTime protocol. For a `DT-Request` packet this field needs to have the value `0x0001`.
- The last 16 bit field 'RequestType' indicates the particular type of request the client makes. When it contains the value `0x0001` the client asks for a textual representation of the current date, when it contains the value `0x0002` the client asks for a textual representation of the current time of day. Other values are not allowed.

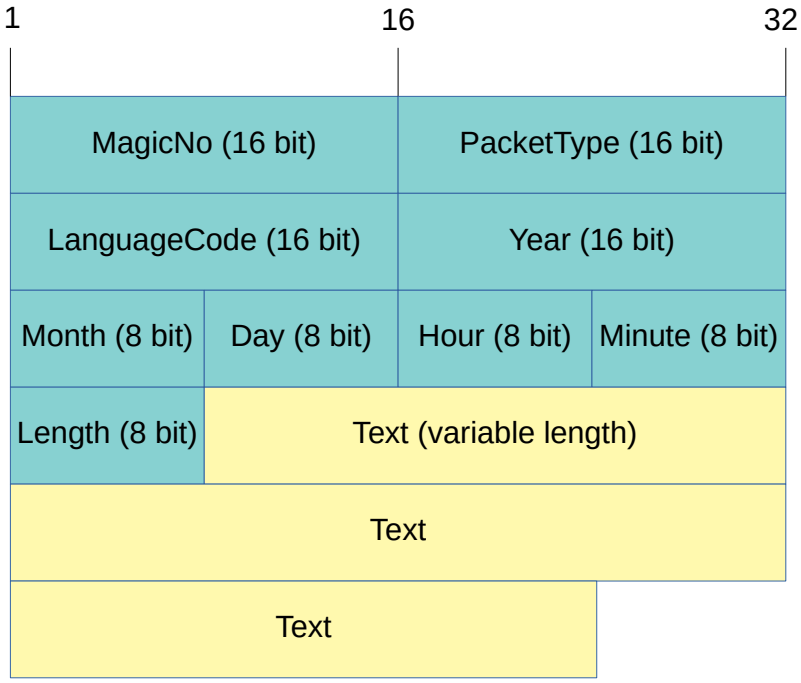
The `DT-Request` packet is being sent from the client to the server. Upon receiving such a packet the server will have to check:

- Whether the packet contains exactly six bytes of data.
- If it does, check whether the 'MagicNo' field contains the value `0x497E`.
- If it does, check whether the 'PacketType' field contains the value `0x0001`.
- If it does, check whether the 'RequestType' field contains either the value `0x0001` or `0x0002`.

If all these conditions are met the server will accept the packet for further processing (see below). If any of these conditions is not met, the server shall simply discard the packet without further action (but printing an appropriate error message on the terminal). Note that this in particular implies that the server will only process received packets with 'PacketType' being equal to `0x0001`, i.e. it processes only `DT-Request` packets.

3.1.2 `DT-Response` packet

The format of the `DT-Response` packet is shown in the following figure:



The first 13 bytes (shown in green) are the fixed header fields. The `DT-Response` packet consists of the following fields:

- The first 16 bit field 'MagicNo' is taken up by a magic number, which again needs to have the value `0x497E`.
- The 16 bit field 'PacketType' indicates the packet type within our DateTime protocol. For a `DT-Response` packet this field needs to have the value `0x0002`.
- The 16 bit field 'LanguageCode' indicates the language used for the textual representation. The value `0x0001` indicates English, the value `0x0002` indicates Te reo Maori, and the value `0x0003` indicates German.
- The 16 bit field 'Year' contains the value for the year as a non-negative integer. As an example, for the current year 2020 this field would contain the value 2020.
- The eight bit field 'Month' contains the value for the month as a non-negative integer: 1 for January, 2 for February, and so on.
- The eight bit field 'Day' contains the value for the day of month as a non-negative integer. This number is allowed to range from 1 to 31.
- The eight bit field 'Hour' contains the hour of the day. This number is allowed to range from 0 to 23.
- The eight bit field 'Minute' contains the minute within the hour. This number is allowed to range from 0 to 59.
- The eight bit field 'Length' indicates the length of the following textual representation in bytes. For example if the text is "hello" (without the speech marks) then the 'Length' field wound contain the value 5.
- The 'Text' field is of variable length and contains the actual text being returned (a textual representation of either the date or the current time of day, see Section 3.4).

The `DT-Response` packet is being sent from the server to the client. Upon receiving such a packet the client will have to check:

- Whether the packet contains at least 13 bytes of data (i.e. all fixed header fields are present).
- If it does, check whether the 'MagicNo' field contains the value `0x497E`.
- If it does, check whether the 'PacketType' field contains the value `0x0002`.
- If it does, check whether the 'LanguageCode' field contains either of the values `0x0001`, `0x0002`, or `0x0003`.
- If it does, check whether the year is a number below 2,100.
- If it is, check whether the Month is a number from 1 to 12.
- If it is, check whether the Day is a number from 1 to 31.
- If it is, check whether the Hour is a number from 0 to 23.
- If it is, check whether the Minute is a number from 0 to 59.

- If it is, check whether the actual length of the entire packet you have received equals the sum of 13 (for the fixed header) plus the contents of the 'Length' field.

If all these conditions are met the client will accept the packet for further processing. If any of these conditions is not met, the client shall simply discard the packet, print an appropriate diagnostic message and exit the program. Note that this in particular implies that the client will only process received packets with 'PacketType' being equal to `0x0002`, i.e. it processes only `DT-Response` packets.

3.2 Server

The server program is started on a host and takes three different port numbers as parameters on the command line. All three port numbers must be different, and they must be numbers between 1,024 and 64,000. If these conditions are not met then the server should exit with an appropriate error message. The first port number is used by clients wanting to get the date/time information in English, the second port number by clients wanting it in Te reo Maori, and the third port number by clients wanting it in German.

Next the server will attempt to open three UDP / datagram sockets and to bind these three sockets to the three given port numbers. If this fails, then the server will exit with an appropriate error message.

After binding the port numbers the server will enter an infinite loop. In this loop:

- The server uses the `select()` system call to wait for a request packet on any of the three sockets. Note that the server should not use any CPU resources while waiting.
- When a packet is received on any of these three sockets, the server:
  - Retrieves the packet from the socket using the Python equivalent of `recvfrom()`. The received packet is stored in a bytearray, and furthermore the server stores the IP address and port number of the packet sender and memoizes on which of the three sockets the packet has been received – this socket determines the language to be used for the textual representation and is also the socket through which the response packet needs to be sent.
  - Performs all the checks necessary to see whether the packet is a valid `DT-Request` packet (see Section 3.1.1). If it is not, print a diagnostic message on the terminal, discard the packet and continue at the start of the loop.
  - Determines from the request whether the client wants to know today's date or the current time of day.
  - Uses a system call to obtain the current time of day $hh : mm$ and date $yyyy : mm : dd$.
  - Prepares a `DT-Response` packet as follows:
    - Prepare the textual representation T of either the date or the time of day in the chosen language as a bytearray (see Section 3.4). Let $|T|$ be the length of the text in bytes (e.g. $|\text{hello}| = 5$). Check whether $|T| \leq 255$. If not, print a diagnostic message and return to the start of the loop without any further processing of this request.
    - Allocate a sufficiently large buffer for the `DT-Response` packet which can hold both the fixed fields and the text T.
    - Fill in all the fixed header fields (from 'MagicNo' to 'Minute') according to the specifications given in Section 3.1.2.
    - Fill in the 'Length' field with $|T|$.
    - Fill in the textual representation T.
  - Finally sends this response packet back to the original sender through the socket on which the corresponding request packet has been received. You will have to use the Python equivalent of `sendto()`, using the client IP address and port number retrieved at the start.

3.3 Client

The client program is started on the same host or another host, and takes three parameters on the command line:

- The first is either the string "date" or the string "time", letting the user specify what he/she wishes to see. If the first parameter is not equal to either of these, the client program should print an error message and stop.
- The second parameter is either an IP address in dotted-decimal notation (e.g. "130.66.22.212") or the host-

name of the computer running the server (e.g. "datetime.mydomain.nz"). The client will attempt to convert this parameter to an IP address using the Python equivalent of the `getaddrinfo()` function. If this conversion fails (e.g. because the hostname does not exist or an IP address given in dotted-decimal notation is not well-formed) then the client should print an error message and stop.

- The third parameter is the port number to use on the server. The port number should be between 1,024 and 64,000. If it is not, then the client should print an error message and stop.

After all command line parameters have been checked the client performs the following operations:

- It opens a UDP socket and prepares a `DT-Request` packet according to the format specified in Section 3.1.1.
- This packet is then sent to the server using the Python equivalent of the function `sendto()`, using the IP address and port number obtained from the parameters as the destination.
- After sending the packet the client should wait for a response packet, but only for a limited time of one second (you may want to look into the Python equivalent of `select()` to limit your waiting time on a socket to one second). If no response packet arrives within one second, the client should print an error message and exit.
- If a response packet arrives within one second, the client should retrieve it from the socket and check whether it is a proper `DT-Response` packet (see Section 3.1.1). If it is not, the client should print an error message and exit.
- The client prints the complete contents of the `DT-Response` packet and then exits.

3.4 Textual Representation

	Date	**Time**
English	Today's date is <name(*mm*)> <*dd*>, <*yyyy*> Today's date is January 7, 2020	The current time is <*hh*>:<*mm*> The current time is 23:22
Te reo Maori	Ko te ra o tenei ra ko <name(*mm*)> <*dd*>, <*yyyy*> Ko te ra o tenei ra ko Kohitatea 7, 2020	Ko te wa o tenei wa <*hh*>:<*mm*> Ko te wa o tenei wa 23:22
German	Heute ist der <*dd*>. <name(*mm*)> <*yyyy*> Heute ist der 7. Januar 2020	Die Uhrzeit ist <*hh*>:<*mm*> Die Uhrzeit ist 23:45

Table 1: Proposed textual representations (including examples)

English	**Te reo Maori**	**German**
January	Kohitātea	Januar
February	Hui-tanguru	Februar
March	Poutū-te-rangi	März
April	Paenga-whāwhā	April
May	Haratua	Mai
June	Pipiri	Juni
July	Hōngongoi	Juli
August	Here-turi-kōkā	August
September	Mahuru	September
October	Whiringa-ā-nuku	Oktober
November	Whiringa-ā-rangi	November
December	Hakihea	Dezember

Table 2: Month names

You should compute a textual representation in the chosen language for both the date and the current time of day. A suggestion for the three chosen languages is given in Table 1. In this representation days, years, minutes and seconds are given as numbers, but a proper name is used for the month (the months names are given in Table 2). Note that if you find it difficult to enter the diacritics (macrons, Umlaut) into the Python source code you can leave those away.

Important point: You should put together the textual representation in Python using normal Python strings. Strings in Python are represented using the UTF-8 character encoding system, and in this system a printed character may require a variable number of bytes to store in memory. Hence, a string which looks like having m characters may in fact need a number of $n \geq m$ of bytes to represent it, and our server program needs to transmit all n bytes. To achieve this, you can use the `encode` method in Python to convert a string to a byte array. For example:

```
...
mystring = "hello world"
```

```
mybytes = mystring.encode('utf-8')
...
```

and then you can use the `len` function to find out the length of the byte array.

3.5 Error Messages and Other Hints

- Appropriate error messages are important that let the user know exactly what happened and what failed. Just having a library function crash on you (possibly printing some nondescript message) does not cut it. We will mark you on that.
- When the client has received a packet, please print it in full, all the fields separately, not as a byte array.
- We discourage the use of append on byte arrays, it makes the code inefficient and harder to read.

4 Deliverables

Each student has to submit **a single pdf file** which includes the following items:

- A cover sheet with your name and student-id.
- A listing of your source code. We would appreciate if you use a pretty printer, i.e. your source code should be shown with some syntax highlighting. This can be achieved from modern IDEs like PyCharm, where you can simply use the function to print a source file – but instead of printing to a printer, you redirect the output to a pdf file. Please do **not** use screenshots!! We will apply deductions if you do.
- You have to print the plagiarism declaration form from the learn page of COSC 264, sign it, scan it, convert the scanned form into a pdf and include this into your submission.

Warning:

- Submissions that are not in pdf format or which contain more than one file automatically receive 0 marks!!
- Submissions which do not include the **signed** plagiarism declaration form automatically receive 0 marks!!

The pdf file has to be submitted via the `learn` page of COSC 264 (see `https://learn.canterbury.ac.nz/course/view.php?id=542&home=1`). Please submit no later than **Sunday, August 16, 2020, 11:59 pm**. Late submissions are **not** accepted, except through the special consideration process.

5 Marking

Marking will be based on the source code. In particular, we will mark it for its ability to produce the right results (e.g. is packet processing correctly implemented, are all the right socket calls there, is the one-second timeout in the client implemented correctly), and for the amount of error / consistency checking you do – if we find that you use system-/socket calls without any error checking or that you do not check incoming packets for correctness, we will apply deductions. We will check whether you have returned all resources (sockets, files) to the operating system. We will also have an eye on style, and may apply deductions if your code is particularly ugly or messy.